**Project-Based Learning Report**

On

**To model PWM generator with variable duty cycle using VHDL and Xilinx simulator**

Submitted in the partial fulfilment of the requirements

For Project-based learning in **VLSI DESIGN TECHNOLOGY**

in

Electronics & Communication Engineering

By

**2114110433 Anshuman**

**2114110449 Ayush Ghosh**

**2114110488 Mayank Mankar**

Under the guidance of the Course In-charge

Prof. Dr. Meena Chavan

Department of Electronics & Communication Engineering

Bharati Vidyapeeth

(Deemed to be University)

College of Engineering,

Pune – 4110043

**Academic Year: 2023-24**

**Bharati Vidyapeeth**

**(Deemed to be University)**

**College of Engineering,**

**Pune – 411043**

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

**CERTIFICATE**

Certified that the Project Based Learning report entitled, “**To model PWM generator with variable duty cycle using VHDL and Xilinx simulator**” is work done by

**2114110433 Anshuman**

**2114110449 Ayush Ghosh**

**2114110488 Mayank Mankar**

in partial fulfilment of the requirements for the award of credits for Project Based Learning (PBL) in **VLSI DESIGN & TECHNOLOGY** of Bachelor of Technology Semester VI, in Electronics and Communication.

**Date: 1 /03/2024**

**Prof. Dr. Meena Chavan Dr. Arundhati A. Shinde**

**Course In-charge Professor & Head E.C.E**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Title** | **Page No.** |
| 1 | problem statement | 1-4 |
| 2 | Software used | 5-6 |
| 3 | Working | 7-11 |
| 4 | VHDL code to implement PWM filter | 12-13 |
| 5 | Algorithm | 14 |
| 6 | Result of simulation | 15 |
| 7 | Conclusion |  |
| 8 | Project Outcome |  |

**PROBLEM STATEMENT**

To model PWM generator with variable duty cycle using VHDL with Xilinx (Vivado) simulator

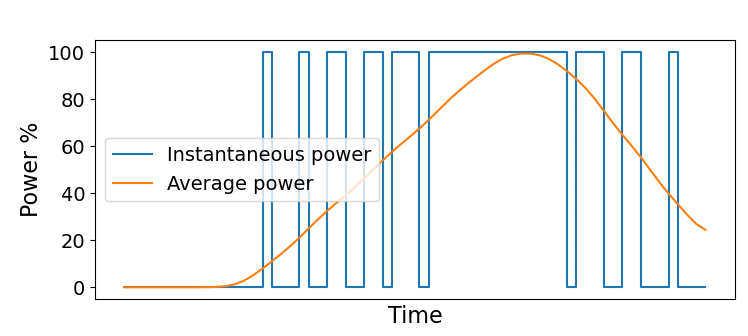
**What is PWM?**

Pulse-width modulation (PWM) is an efficient way to control analog electronics from purely digital FPGA pins. Instead of attempting to regulate the analog voltage, PWM rapidly switches on and off the supply current at full power to the analog device. This method gives us precise control over the moving average of energy provided to the consumer device.

Examples of use cases that are good candidates for PWM are audio modulation (speakers), light intensity control (lamps or LEDs), and induction motors. The latter includes servo motors, computer fans, pumps, brushless DC motors for electric cars, and the list goes on.

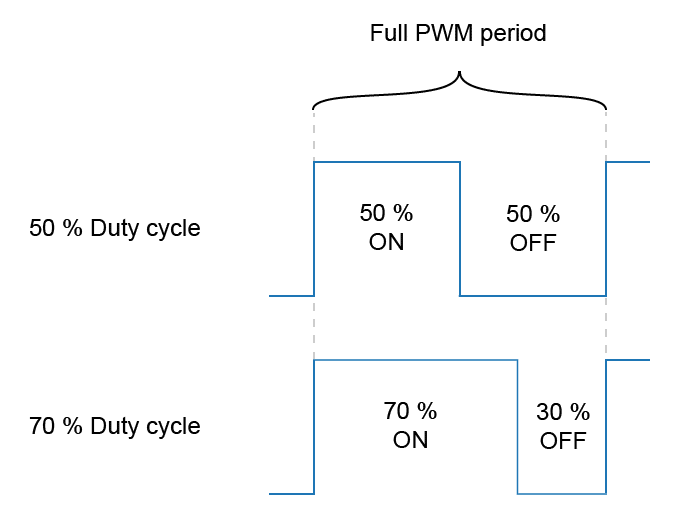
## **How PWM works**

## **By switching on and off the power supply to a device with a high frequency, we can accurately control the average current flowing through it. The illustration below shows the basics of how PWM works. The PWM output controls a binary switch that can either set the power to 100% or 0%. By quickly alternating between the two extremes, the sliding window average will be a function of the time spent in each of the states.**



**Duty cycle**

The duty cycle is key to controlling the power given to the analog device in PWM. The term duty cycle means how much time the PWM output spends at the ON position. It’s common to describe the duty cycle as a percentage, as shown in the image below. However, in my VHDL example, I will use an unsigned binary number later in this article. It makes more sense for us to use a binary number, which can represent the full resolution of the duty cycle in our VHDL implementation.



With a duty cycle of 0, the PWM output would remain at the OFF position continuously, while at 100%, it would be non-stop at the ON position. The degree of accuracy that the PWM controller can exert on the payload effect is directly related to the length of the PWM counter. We shall see how this works in the VHDL code when we implement a PWM controller later in this article.

The formula to convert the binary representation of the duty cycle to a percentage is shown below.

## **A close-up of a white background Description automatically generated**

**PWM frequency**

When talking about PWM switching frequency, we mean how often the PWM output alternates between the ON and OFF states, how long it takes for the PWM counter to wrap. As always, the frequency is the inverse of the full PWM period:

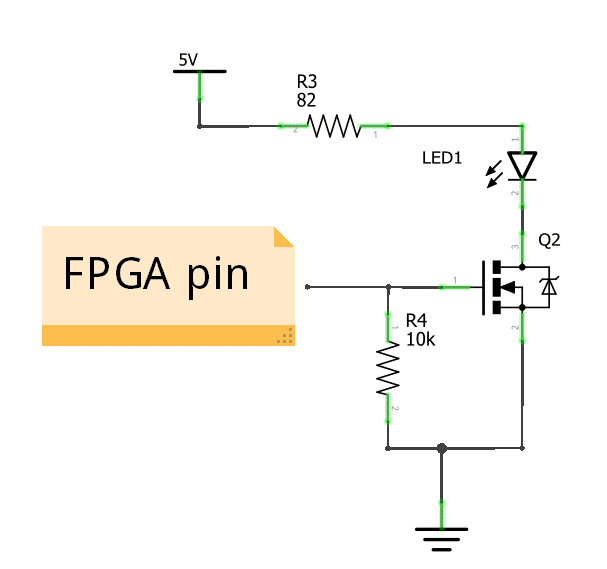
A mathematical equation with black text

Description automatically generated

The ideal PWM frequency depends on what kind of device you are controlling. Any number larger than a few hundred Hertz will look like a stable light source to the naked eye if the consumer is an LED. For a brushless DC motor, the sweet spot lies in the tens of kilohertz range. Set the frequency too low, and you may experience physical vibrations. With a too fast oscillation, you are wasting power.

An issue to keep in mind is that the analog power electronics isn’t as fast as the digital FPGA pin. A typical PWM setup uses power MOSFETs as switches to control the current flowing through the analog device.

Consider the schematic shown in the image. It’s part of the LED driver circuit used in my advanced Dot Matrix VHDL course. The FPGA pin controls the gate of the MOSFET, acting as a circuit breaker to the in-series LED. With a higher switching frequency, the transistor will spend more time not being entirely open nor fully closed. That translates into wasted power and excess heat production in the MOSFET.



**SOFTWARE USED**

**Logo

Description automatically generated with medium confidenceXILINX VIVADO**

Vivado Design Suite is a software suite produced by Xilinx for the synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system-on-a-chip development and high-level synthesis. Vivado represents a ground-up rewrite and rethinking of the entire design flow (compared to ISE). Like the later versions of ISE, Vivado includes the in-built logic simulator. Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic

**FEATURES**

Vivado was introduced in April 2012,[1] and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards-based packaging of both algorithmic and RTL IP for reuse; standards-based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.[13] A free version of WebPACK Edition of Vivado provides designers with a limited version of the design environment.[14]

**COMPONENTS**  
The Vivado High-Level Synthesis compiler enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading. Vivado 2014.1 introduced support for automatically converting OpenCL kernels to IP for Xilinx devices. OpenCL kernels are programs that execute across various CPU, GPU and FPGA platforms.

The Vivado Simulator is a component of the Vivado Design Suite. It is a compiled-language simulator that supports mixed-language, Tcl scripts, encrypted IP and enhanced verification.

The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library. The Integrator is also tuned for MathWorks Simulink designs built with Xilinx's System Generator and Vivado High-Level Synthesis.

**Logo

Description automatically generated**

**WORKING**

VHDL (VHSIC Hardware Description Language) is used to describe the behavior of the PWM generator.The VHDL code typically consists of an entity declaration and an architecture definition.

Within the architecture, components such as counters, debouncers, and control logic are instantiated and interconnected to form the PWM generator circuit.

Variable Duty Cycle:

The PWM signal's duty cycle represents the proportion of time the signal is in the high state compared to the total period.

In this project, the duty cycle is made adjustable, meaning it can be changed during operation.

This is usually achieved by incorporating control logic that responds to external inputs (such as buttons or switches) to increase or decrease the duty cycle.

Debouncing:

External inputs (e.g., buttons) used to control the duty cycle may suffer from bouncing, where the signal oscillates between high and low states rapidly when pressed or released.

To ensure reliable operation, debouncing techniques are applied to filter out these rapid transitions and produce clean, stable control signals.

This may involve using flip-flops or other debounce circuits to detect and respond to the stable state of the input signal.

Xilinx Simulator

Xilinx provides simulation tools such as ISim (for ISE) or Vivado Simulator (for Vivado) to verify the functionality of the VHDL design.

The VHDL code for the PWM generator, along with any necessary testbench code, is compiled and simulated using these tools.

During simulation, the behavior of the PWM signal and other internal signals within the circuit are monitored and analyzed over time.

Simulation Process:

The simulation process involves applying stimulus to the inputs of the PWM generator, such as clock signals and control signals.

The simulation tool simulates the propagation of these signals through the VHDL circuit and tracks the resulting behavior of the PWM signal.

Engineers can observe the waveform of the PWM signal and analyze its characteristics, including the duty cycle and frequency.

Verification and Debugging:

During simulation, engineers verify that the PWM generator operates as expected, producing a variable duty cycle in response to control inputs.

Any unexpected behavior or errors are identified and debugged by examining the simulation results and adjusting the VHDL code as necessary.

Documentation and Reporting:

The simulation results, along with any design constraints, optimizations, or modifications, are documented.

Reports may be generated to summarize the simulation findings and validate the design against requirements.

**VHDL code to model PWM generator with variable duty cycle using VHDL with Xilinx (Vivado) simulator**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity PWM\_Generator is

port (

clk: in std\_logic; -- 100MHz clock input

DUTY\_INCREASE: in std\_logic; -- button to increase duty cycle by 10%

DUTY\_DECREASE: in std\_logic; -- button to decrease duty cycle by 10%

PWM\_OUT: out std\_logic -- PWM signal out with frequency of 10MHz

);

end PWM\_Generator;

architecture Behavioral of PWM\_Generator is

-- D-Flip-Flop for debouncing module

component DFF\_Debounce

Port (

CLK : in std\_logic;

en : in std\_logic;

D : in std\_logic;

Q : out std\_logic

);

end component;

signal slow\_clk\_en: std\_logic:='0'; -- slow clock enable for debouncing

signal counter\_slow: std\_logic\_vector(27 downto 0):=(others => '0');-- counter for creating slow clock

signal tmp1,tmp2,duty\_inc: std\_logic;-- temporary signals for deboucing

signal tmp3,tmp4,duty\_dec: std\_logic;-- temporary signals for deboucing

signal counter\_PWM: std\_logic\_vector(3 downto 0):=(others => '0');-- counter for PWM signal

signal DUTY\_CYCLE: std\_logic\_vector(3 downto 0):=x"5";

begin

-- Debouncing process

-- First generate slow clock enable for deboucing (4Hz)

process(clk)

begin

if(rising\_edge(clk)) then

counter\_slow <= counter\_slow + x"0000001";

--if(counter\_slow>=x"17D7840") then -- for running on FPGA -- comment when running simulation

if(counter\_slow>=x"0000001") then -- for running simulation -- comment when running on FPGA

counter\_slow <= x"0000000";

end if;

end if;

end process;

--slow\_clk\_en <= '1' when counter\_slow = x"17D7840" else '0';-- for running on FPGA -- comment when running simulation

slow\_clk\_en <= '1' when counter\_slow = x"000001" else '0';-- for running simulation -- comment when running on FPGA

-- debounce part for duty increasing button

stage0: DFF\_Debounce port map(clk,slow\_clk\_en , DUTY\_INCREASE, tmp1);

stage1: DFF\_Debounce port map(clk,slow\_clk\_en , tmp1, tmp2);

duty\_inc <= tmp1 and (not tmp2) and slow\_clk\_en;

-- debounce part for duty decreasing button

stage2: DFF\_Debounce port map(clk,slow\_clk\_en , DUTY\_DECREASE, tmp3);

stage3: DFF\_Debounce port map(clk,slow\_clk\_en , tmp3, tmp4);

duty\_dec <= tmp3 and (not tmp4) and slow\_clk\_en;

-- for controlling duty cycle by these buttons

process(clk)

begin

if(rising\_edge(clk)) then

if(duty\_inc='1' and DUTY\_CYCLE <= x"9") then

DUTY\_CYCLE <= DUTY\_CYCLE + x"1";--increase duty cycle by 10%

elsif(duty\_dec='1' and DUTY\_CYCLE>=x"1") then

DUTY\_CYCLE <= DUTY\_CYCLE - x"1";--decrease duty cycle by 10%

end if;

end if;

end process;

-- Create 10MHz PWM signal

process(clk)

begin

if(rising\_edge(clk)) then

counter\_PWM <= counter\_PWM + x"1";

if(counter\_PWM>=x"9") then

counter\_PWM <= x"0";

end if;

end if;

end process;

PWM\_OUT <= '1' when counter\_PWM < DUTY\_CYCLE else '0';

end Behavioral;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity DFF\_Debounce is

Port (

CLK : in std\_logic;

en: in std\_logic;

D : in std\_logic;

Q : out std\_logic

);

end DFF\_Debounce;

architecture Behavioral of DFF\_Debounce is

begin

process(CLK)

begin

if (rising\_edge(CLK)) then

if (en='1') then

Q <= D;

end if;

end if;

end process;

end Behavioral;

**Testbench code for PWM filter**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY tb\_PWM\_Genenrator IS

END tb\_PWM\_Genenrator;

ARCHITECTURE behavior OF tb\_PWM\_Genenrator IS

COMPONENT PWM\_Generator

PORT(

clk : IN std\_logic;

DUTY\_INCREASE : IN std\_logic;

DUTY\_DECREASE : IN std\_logic;

PWM\_OUT : OUT std\_logic

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal DUTY\_INCREASE : std\_logic := '0';

signal DUTY\_DECREASE : std\_logic := '0';

--Outputs

signal PWM\_OUT : std\_logic;

-- Clock period definitions

constant clk\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: PWM\_Generator PORT MAP (

clk => clk,

DUTY\_INCREASE => DUTY\_INCREASE,

DUTY\_DECREASE => DUTY\_DECREASE,

PWM\_OUT => PWM\_OUT

);

-- Clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

DUTY\_INCREASE <= '0';

DUTY\_DECREASE <= '0';

wait for clk\_period\*10;

DUTY\_INCREASE <= '1';

wait for clk\_period\*10;

DUTY\_INCREASE <= '0';

wait for clk\_period\*10;

DUTY\_INCREASE <= '1';

wait for clk\_period\*10;

DUTY\_INCREASE <= '0';

wait for clk\_period\*10;

DUTY\_INCREASE <= '1';

wait for clk\_period\*10;

DUTY\_INCREASE <= '0';

wait for clk\_period\*10;

DUTY\_DECREASE <= '1';

wait for clk\_period\*10;

DUTY\_DECREASE <= '0';

wait for clk\_period\*10;

DUTY\_DECREASE <= '1';

wait for clk\_period\*10;

DUTY\_DECREASE <= '0';

wait for clk\_period\*10;

DUTY\_DECREASE <= '1';

wait for clk\_period\*10;

DUTY\_DECREASE <= '0';

wait for clk\_period\*10;

wait;

end process;

END;

**ALGORITHM**

1.Clock Generation:

Generate a clock signal of 100MHz (clk) for the entire system.

2.Debouncing:

Generate a slow clock signal at a frequency of 4Hz for debouncing purposes (slow\_clk\_en).

Implement a DFF-based debouncing module (DFF\_Debounce) to eliminate button bounce.

Use this debounced signal to capture button presses (DUTY\_INCREASE and

DUTY\_DECREASE).

3.Duty Cycle Control:

Use debounced button signals (duty\_inc and duty\_dec) to increase or decrease the duty cycle.

Increment or decrement the duty cycle by 10% accordingly.

4. PWM Generation:

Generate a 10MHz PWM signal (PWM\_OUT) using a counter (counter\_PWM) to control the

PWM frequency.

Compare the counter value with the duty cycle to determine the PWM output state.

5.Main Processes:

Synchronize processes to the rising edge of the clock (rising\_edge(clk)).

Increment the slow clock counter (counter\_slow) to generate the slow clock enable signal

(slow\_clk\_en).

Increment the PWM counter (counter\_PWM) to generate the PWM signal.

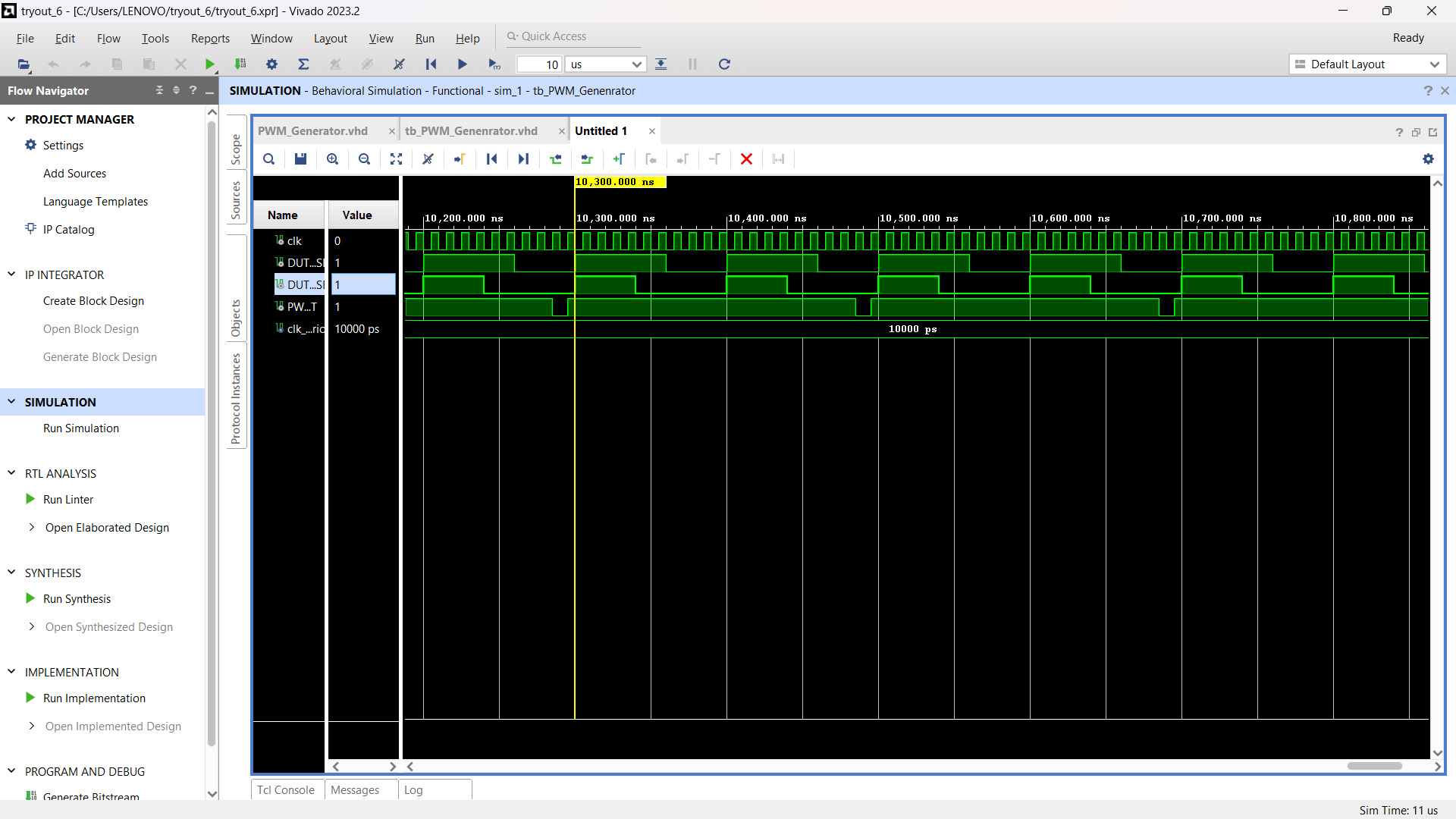
6.Signal Routing:

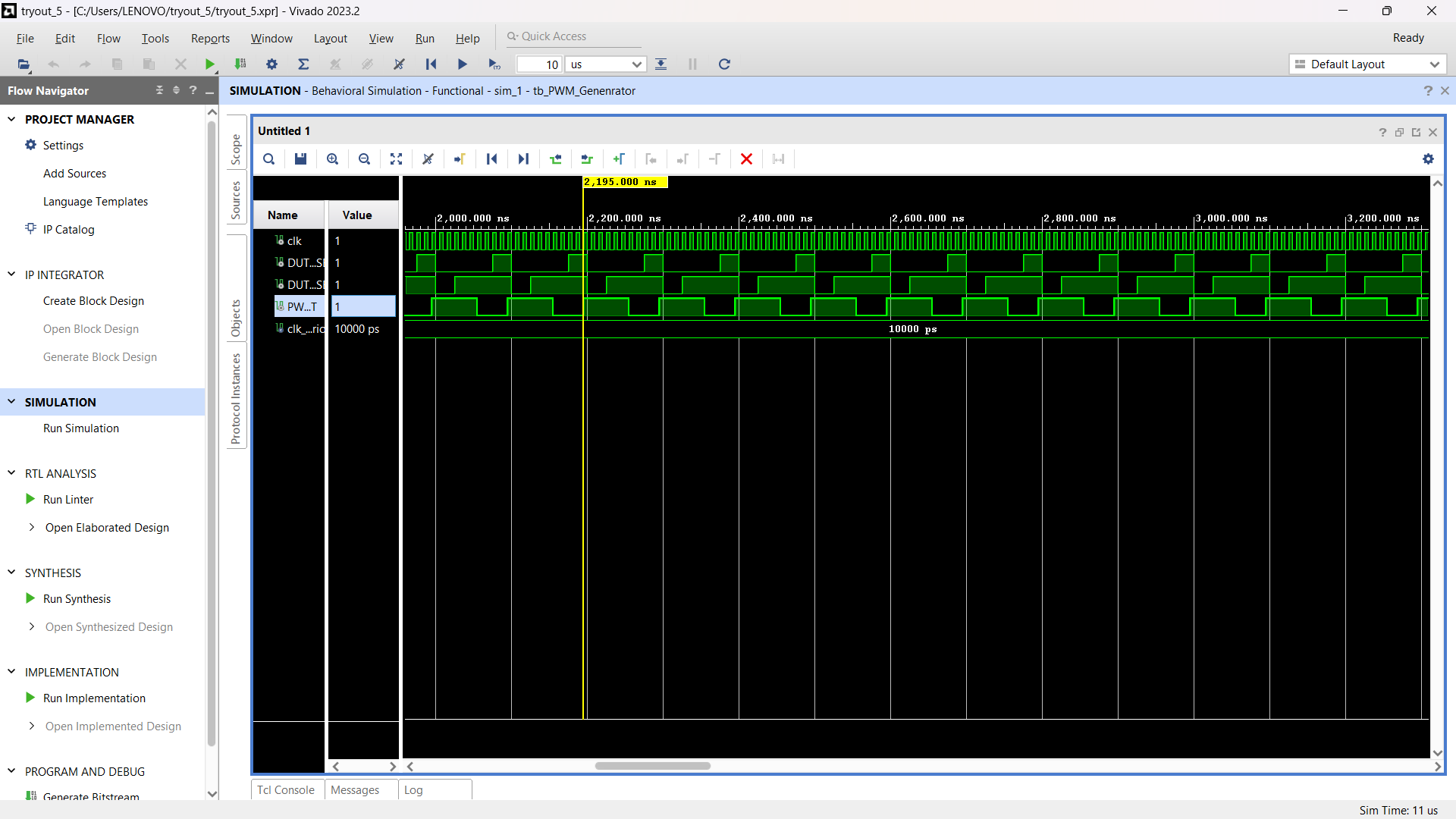
Connect signals and ports according to the entity declaration.

Use intermediate signals (tmp1, tmp2, tmp3, tmp4) for debouncing and control logic.

Route the final PWM signal to the output port (PWM\_OUT).

**RESULT OF SIMULATION**





**CONCLUSION**

Hence, in this Project-Based Learning, on the topic- ‘To model PWM generator with variable duty cycle using VHDL and Xilinx simulator’, we have learned the concepts of VLSI design technology, and we have understood the fundamental programming of VHDL. All the related concepts were understood well and were performed in the software and simulated successfully.

**COURSE OUTCOME**

Here, in this Project-Based Learning, under Course Outcome 1 (CO1), Designing and simulating a digital system using Structural, Behavioral, Dataflow, or Mixed style of Modelling was developed and evaluated. The concepts were understood and performed practically using XILINX.

**REFERENCES**

* <https://www.fpga4student.com/2017/08/car-parking-system-in-vhdl-using-FSM.html>
* [**digital-systems-design-with-vhdl-labs**](https://github.com/marcelcases/digital-systems-design-with-vhdl-labs/tree/master)/[Lab1\_PWM](https://github.com/marcelcases/digital-systems-design-with-vhdl-labs/tree/master/Lab1_PWM)
* "Digital Design and Computer Architecture" by David Money Harris and Sarah L. Harris. This book covers digital design concepts including PWM generation.
* "VHDL for Digital Design" by Frank Vahid and Roman Lysecky. It offers a comprehensive guide to VHDL for digital design, including PWM generation techniques.